Artificial Mutation inspired Hyper-heuristic for Runtime Usage of Multi-objective Algorithms

Donia El Kateb, François Fouquet, Johann Bourcier, Yves Le Traon

Sputnik: Elitist Artifical Mutation Hyper-heuristic for Runtime Usage of Multi-objective Evolutionary Algorithms

Donia El Kateb and François Fouquet
and Yves Le Traon
SnT - University of Luxembourg
first.last@uni.lu

Johann Bourcier
University of Rennes 1 / IRISA / INRIA
johann.bourcier@inria.fr

ABSTRACT

In the last years, multi-objective evolutionary algorithms (MOEA) have been applied to different software engineering problems where many conflicting objectives have to be optimized simultaneously. In theory, evolutionary algorithms feature a nice property for runtime optimization as they can provide a solution in any execution time. In practice, based on a Darwinian inspired natural selection, these evolutionary algorithms produce many deadborn solutions whose computation results in a computational resources wastage: natural selection is naturally slow. In this paper, we reconsider this founding analogy to accelerate convergence of MOEA, by looking at modern biology studies: artificial selection has been used to achieve an anticipated specific purpose instead of only relying on crossover and natural selection (i.e., Muller *et al* [18] research on artificial mutation of fruits with X-Ray). Putting aside the analogy with natural selection , the present paper proposes an hyper-heuristic for MOEA algorithms named *Sputnik* [1] that uses artificial selective mutation to improve the convergence speed of MOEA. *Sputnik* leverages the past history of mutation efficiency to select the most relevant mutations to perform. We evaluate *Sputnik* on a cloud-reasoning engine, which drives on-demand provisioning while considering conflicting performance and cost objectives. We have conducted experiments to highlight the significant performance improvement of Sputnik in terms of resolution time.

Keywords

MOEA, Hyper-heuristic, Artificial mutation, Optimization, Cloud

1. INTRODUCTION

Beyond well known harmful computer viruses, Computer Science has also often been inspired by biological processes for more constructive purposes. For example, John Holland was inspired by the Darwinian evolution [2] to design genetic algorithms [13, 12]. These algorithms are particularly efficient to solve complex problems which have very large solution spaces. Since their introduction in 1974, they have been applied in numerous contexts (biology, computer science, mathematics) for simulation, optimal allocation, function optimisation. Multiple-objective evolutionary optimization (MOEA) [3, 21, 3] is thus a domain that tackles a category of problems in which a decision maker aims at finding a solution that optimizes several conflicting objectives. MOEA are used nowadays in several design case studies [7] like cloud organization optimization, however their applicability at run-time presents several issues related to performance. MOEA algorithms leverage an evolution principle based on Darwinian [19] rules that derives, from an initial population of solutions, new solutions combining acceptable trade-offs between objectives. Therefore, MOEA algorithms mimic the natural selection by leveraging a set of fitness functions to evaluate a solution to a specific problem. Additionally, they combine a set of operators [20] to drive population evolution by introducing small changes on individuals (i.e, mutation), composing new ones (i.e, crossover) or selecting the candidates that will compose next generations (i.e, selection). However, natural selection is based on a random mutation in order to mimic the equity of species expressed in Darwinian rules. The result of this algorithm is the genotype that is the closest to the optimal solution for the problem. However, this randomness leads to sub-optimal performance for multiple-objective resolution mainly because of the creation of several unused generations wasting computational resources. Besides, modern genetic does not try to mimic the real evolution process in labs. Instead, based on the founding work of Muller *et al* [18], artificial mutation is now widely used to save time and generation cycles for instance to produce genetically modified organisms. Instead of relying only on crossover and natural selection, Muller *et al* [18] studied artificial mutation using X-Ray to modify a fruit with an anticipated intent. These principles have led the genetic field to build instruments for such selective artificial mutation. Going along the same line, we study how such principles from the genetics could be adapted to MOEA to accelerate the convergence to a solution by guiding the evolutionary algorithms through dynamically selected mutation operators.

Our intuition is that operators applied in a smart and artificial way would provide better results than operators applied randomly, and in particular would reduce the number of useless solutions construction. So, the new algorithm we propose is no longer inspired by Darwinian evolution, but by "artificial mutation", based on a smart and dynamic selection of the best mutation operator to apply at a given step. By applying such operators in priority, we aim at orienting the evolution process of a given population in the right direction for the problem to solve. This makes our new algorithm a new category of hyper-heuristic [6] in the sense it aims at improving performance for a specific problem [10] as the search evolves.

In this paper, we present this new hyper-heuristic algorithm, called "Sputnik", inspired by artificial mutation. Its name "Sputnik" from

[1] https://en.wikipedia.org/wiki/Sputnik_virophage

a virus family which evolves and mutates together with their host in order to perfectly fit their environment and to replicate more quickly. In the same manner, Sputnik algorithm leverages a continuous ranking of operators according to their impact on fitness functions to smartly select dynamically the mutation operators according to the problem to solve. We focus on performance as a key factor for runtime usage to reach faster acceptable trade-offs while saving computation time and generation cycles. For instance, the acceleration Sputnik provides is useful for adaptive systems when a solution/reaction has to be found in a short time. We evaluate our approach on a cloud reasoning engine which handles several conflicting objectives (i.e, latency, cost) and is able to continuously provision customers software by different available cloud providers. The validation focuses on the performance of our hyper-heuristic for multi-objective resolution. We have integrated Sputnik in the Polymer framework [2] and evaluated it using a model@runtime platform [3]. We have conducted an experiment to compare natural selection versus Sputnik. Our experiments highlight that Sputnik results in a faster convergence by reducing the number of generations while conserving the ability to achieve acceptable trade-offs on the considered use case. This paper is organized as follows. Section 2 describes the key concepts related to this paper. Section 3 presents Sputnik hyper-heuristic. Section 4 presents validation elements of our approach. Finally, section 5 and 6 discuss the related work, our conclusion and future work.

2. MOEA FOR AT-RUNTIME USAGE

Multi-objective evolutionary algorithms (MOEA) are optimization algorithms driven by elitism rules that favor the survival of strongest species [20] in analogy to natural selection. Applied to software engineering, species are candidate solutions to optimization problems which constitute what is called a *generation*. MOEA are based on an iterative search in which a set of individuals is selected and mutated in each iteration to constitute a new *generation*.

Fitness functions are used to evaluate solutions to solve a specific optimization problem, in analogy to natural selection where species qualities are evaluated according to their surrounding context. Genetic operators: *crossover* and *mutation* [20] are used to emulate natural mutation in order to generate new *generations* composed of mutated solutions. Mutation operators introduce small changes with a small probability to preserve the population diversity [16] whereas crossover operators create new individuals by genetic recombination. MOEA algorithms can be used to solve at runtime optimization problems, for instance a load balancing problem, therefore performance becomes a major issue to consider [14]. Among the contributions that have addressed these issues, hyper-heuristics [6] are self-tuning heuristics that are based on parameters adaptation during optimization processing. In general, hyper-heuristics introduce modifications on the algorithm itself, in order to improve its efficiency to handle a specific purpose [10].

The contribution of this paper, detailed in section 3, falls into the category of hyper-heuristics that embed learning mechanisms to achieve performance improvement over a set of software engineering problems [9, 17]. It relies on an heuristic selection mechanism that evaluates search parameters during the search process and grants higher priority for the most effective ones.

3. SPUTNIK

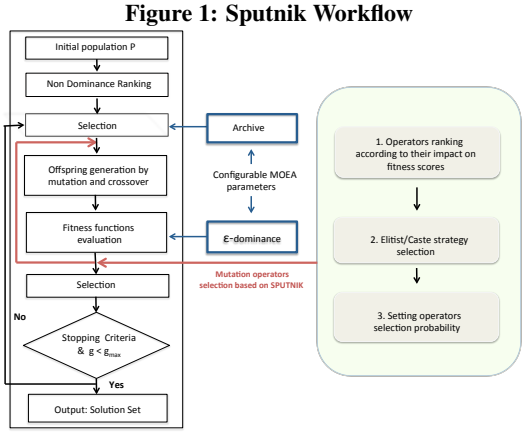
The randomness introduced by natural selection of evolutionary approaches leads to suboptimal performance in terms of computational power and memory usage. Random selection of mutation operators produces useless candidate solutions that lead to computational resources waste and therefore does not meet run-time optimization constraints. In modern biological studies, after identifying a gene impact on individual phenotype trait, scientists like Muller *et al* [18] leverage artificial mutation to directly produce an individual combining the foreseen modification.

Our hypothesis is that the artificial mutation concept can be introduced in evolutionary algorithms to mimic modern biological genetic, thus reducing the number of required generations to reach acceptable solutions. The *a priori* scientific knowledge of a gene modification impact, could be replaced by a continuous ranking and learning approach leveraging execution history. Therefore, in this paper we aim at optimizing MOEA, by dynamically reducing the usage of mutation operators that are less effective in improving fitness functions scores. At the same time, we maintain the equity of natural selection, to ensure that the modified evolution algorithm is able to reach any solution. Thus, we replace the random mutation operator selection by an hyper-heuristic that detects for each individual the most pertinent operator to apply in order to achieve a faster trade-off.

After each application, mutation operators are classified according to the delta variance they introduce on each fitness function. Internally, Sputnik maintains an elitist group of mutation operators that are relevant to improve a fitness function score. To enhance operators selection, Sputnik, considers the current fitness scores reached by a solution, and selects the most relevant mutator in elite groups to improve next generation [4].

Sputnik workflow, depicted by the flowchart of Figure 1, takes as inputs an initial population and a generation number. As most of MOEA variants like (NSGA-II, ε-MOEA, SPEA 2 [21]) the major part of the algorithm is an individual ranking step according to

Figure 1: Sputnik Workflow



Algorithm 1 Sputnik

```
<Sputnik (Population-Size int, Nb Generation int)>
for all j where j range from 1 to Generation-Number do
    Evaluate f_i(x) on P new and calculate crowding distance
    Update P
    Apply randomly a set of mutation operators O from O_set on P to get P_new
    operator-used:=operator-used ∪ O
    if operator-used ⊆ O_set then
        evaluate △_impact f_gi ∀ O in O_set
    end if
    select P ∈ {P_elitist, P_caste}
    select O from O_set with P to get P_new with P
end for
```

[2] http://http://kevoree.org/polymer
[3] http://kevoree.org
[4] http://www.genetics.org/content/111/1/147.short

fitness function evaluation and a non dominating population construction. Sputnik introduces a favoritism operator approach in the mutation process.

We consider a multi-objective evolutionary optimization of f with n objectives (f_1, f_2,...,f_n). The average fitness score for a generation g_i is defined by $\sum_{i=1}^{n} f_i/n$. We define $\triangle_{impact} f_{g_i}$ as the fitness score variation between the average of fitness function evaluation for a generation g_{i-1} and a generation g_i: $\triangle_{impact} f_{g_i}$=$(\sum_{i=1}^{n} f_i/n)_{g_i}$-$(\sum_{i=1}^{n} f_i/n)_{g_{i-1}}$. Sputnik records the *selection occurrence* for the different mutation operators that have been involved in each generation. Once all mutators have been selected at least once, $\triangle_{impact} f_{g_i}$ is evaluated for all the operators and Sputnik can be configured to select the operators that have $\triangle_{impact} f_{g_i} > 0$ in the generation g_{i+1} with the two following settings:

- *Elitist Strategy*: The operator that has the highest *delta impact positive* is always chosen in the generation g_{i+1}. This configuration accords higher chance to the "winner operator" to be selected in the next generation.
- *Caste Strategy*: A selection probability is partitioned between operators that have $\triangle_{impact} f_{g_i} > 0$ and is defined as $P_{selection} = \triangle_{impact} f_{g_i} / \sum_{i=1}^{n} \triangle_{impact} f_{g_i}$ This configuration gives more equity in terms of selection probability for all operators which have a positive impact on a fitness.

SPUTNIK-based mutation operators selection is described in Algorithm 1. In both settings a probability of 10% is maintained for pure random selection of operators to not discriminate worst ranked operators. Sputnik keeps 10% of mutation to give chance to less selected operators to be reintroduced in the elite group of a fitness function. By this mechanism we keep a minimal equity of species while conserving 90% of the Pareto for most efficient mutation.

4. VALIDATION

We validate our approach through a reasoning engine for an hybrid cloud management system: A cloud customer aims at hosting some of his applications on a set of virtual machines that are dedicated for his internal usage in a private cloud provider. The customer aims at reducing costs inherent in a setting based on private cloud model by exporting some applications to a public cloud. The problem can be formulated as follows: "Given a dedicated set of VMs allocated by a private cloud provider, how to optimize software placement in VMs to reduce **costs** and to reduce **latency** that can be introduced by a distant hosting?".

We define a cloud configuration as an architecture model which leverages virtual machine and component (i.e, provisioned software in our context) concepts. Based on our architectural model, an *Individual* represents a solution vector $x \in X$ that corresponds to a cloud infrastructure model. A *gene* corresponds to a component or a virtual machine in our model. A *population* corresponds to a set of cloud infrastructure models. A *Genetic mutation operator* corresponds to an elementary flip in the model like *AddComponent(c,A)*. For the sake of space limitations, we will not give ample details about the operators defined in this paper. The reader may refer to our open source Polymer framework which gives ample details about our architectural model [5].

A Cloud infrastructure multi-objective optimization problem is represented by the following Triplet *(I,F,CO)*. I denotes a cloud infrastructure model which represents an abstraction of a set of (VM). Each (VM) hosts n software components (C). *CO* denotes a set of possible configurations in *I* that satisfy F. A configuration $co \in CO$ is obtained through a mapping from components to (VMs), for example $co = (VM_1(c_1, c_2, c_3))$ denotes a configuration with a single virtual machine VM_1 hosting 3 components c_1, c_2, c_3.

[5]https://github.com/dukeboard/kevoree-genetic

The vector F(X) is composed of the following 2 objective functions, $F(X) = (f_1(x), f_2(x))$ that have to be minimized: 1) $f_1(x)$= Cost(x): denotes the cost which is proportional to the number of active VMs. 2) $f_2(x)$= Latency(x): The latency function is calculated on the basis of the average of latency of all components in the model, considering both redundancy and distant hosting impacts.

The multi-objective optimization problem aims at finding a configuration $co \in CO$ such as $\min_{co} F(X)$. We have implemented Sputnik prototype in the Polymer framework which leverages a model based encoding to perform MOEA optimization. Indeed, using model@run.time paradigm [1], our cloud models can be seamlessly deployed in a real large-scale production environment [8] like a cloud infrastructure. The evaluation of cloud Sputnik based reasoning engine is out of scope of this paper, only the gain achieved by the hyper-heuristic will be evaluated in this paper.

4.1 Research Questions

This validation section aims at exploring the effectiveness of Sputnik to reduce the generations number to reach a certain level of trade-off. Secondly, we provide evidence that the results are comparable in terms of level of trade-off achieved.

Finally, we explore the effectiveness of Sputnik once embedded in most popular MOEA algorithms such as MOEA-D and NSGA-II. To compare the solutions of the different algorithms under study, we choose the hypervolume metric as Pareto-front quality indicator [11]. Our validation steps can be summarized in the following research questions: RQ_1) Considering that an acceptable trade-off is 90% of the best solutions, is the Darwin Sputnik operator selection strategy successful to reduce the number of generations to reach the acceptable trade-off compared to a classical random strategy? RQ_2) Does Sputnik produce comparable results in terms of trade-off achieved regarding classical random mutation selection? RQ_3) Is Sputnik relevant for common MOEA algorithms?

4.2 Experimental results

To answer RQ_1, we run Sputnik in a cloud optimization engine which contains 100 virtual nodes and several components that have to be dispatched. Our cloud reasoning engine is configured with one of the most popular MOEA algorithm NSGAII [4]. The experiment is run 3 times for 300 generations, Sputnik with caste strategy, Sputnik with elitist strategy and finally with random. The results are shown in the Figure 2. We consider that a solution achieves an acceptable trade-off if it reaches 90% of the best obtained solution. In our case study, the best obtained solution achieves an hyper-volume of 0.79 (acceptable hypervolume value is 0.71 in our case), it has been reached with a 2000 generations previous run. A run with Sputnik (with both strategies) reaches this value after 176 generations whereas a run with random operators selector reaches this value after 279 generations. Sputnik strategies are very similar in terms of hypervolume achievements. We notice that elitist converges slighly faster however, the caste strategy can reach better hyper-volume scores because it takes benefit from mutators diversity. In average, Sputnik (both with cast and elitist) outperforms random selector by reducing around 37% the number of necessary generations. This confirms our first hypothesis which states that a smart mutators selection strategy is successful to reduce the number of generations to reach an acceptable trade-off compared to a classical random selection strategy.

To reply to RQ_2, we run a similar experiment with both random and Sputnik selector until we reach an unchanged value of hypervolume for 50 generations. Final values for Sputnik (elitist: 0.77 and caste: 0.81) and random (0.78) ($\pm$1% each difference) allow us to conclude that our hyper-heuristic does not decrease the quality

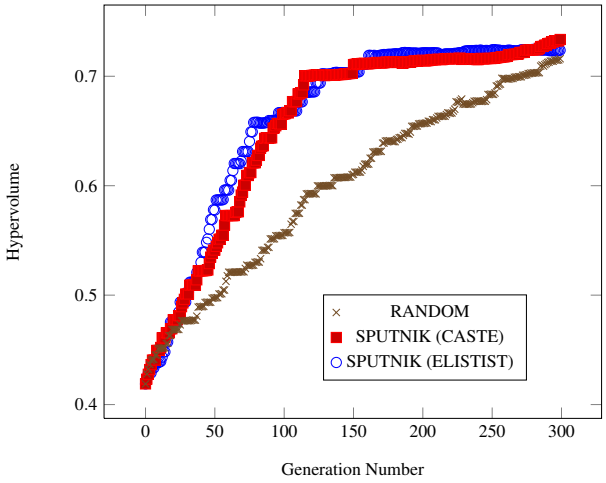


Figure 2: Hypervolume: Sputnik (Elitist & Caste) vs Random

of the result in terms of degree of trade-off achieved. Moreover the caste strategy improves the hypervolume score.

For RQ_3, we set up common MOEA algorithms with Sputnik. The results shown in Table 1, are hypervolume values after 200 generations compared to MOEA without Sputnik. We conclude that Sputnik (both with caste and elitist configurations) is effective in achieving a better trade-off faster than current MOEA.

	NSGAII	ε-NSGA II	ε-MOEA	Hyper-MOEA
Elitist	**0.72**	**0.75**	**0.51**	**0.47**
Caste	**0.70**	**0.73**	**0.52**	**0.47**
Random	**0.67**	**0.66**	**0.43**	**0.44**

Table 1: Sputnik (Elitist & Caste) hypervolume on MOEA

5. RELATED WORK

Several approaches have proposed hyper-heuristics that operate on top of MOEA to improve their run-time usage. In [10], the authors embed learning techniques in classical MOEA. They assume that objective functions are expensive to compute so they rank the pareto front elements and they evaluate only the individuals that have higher ranks. In [15], the authors have proposed an hyper-heuristic that relies on the hypervolume calculation to improve computational results. These approaches consider only the Pareto front set evaluation to improve MOEA, whereas our approach evaluates operators contribution in improving fitness and thus injects mutation operators that are eligible to make MOEA converge faster. In [22], the authors have shown that racing algorithms can be used to reduce the computational resources inherent from using evolutionary algorithms in large scale experimental studies, their approach automates solutions selection and discards solutions that can not come up with results improvement. Whereas racing techniques eliminate worst solutions candidates to speed up the search, in our approach we keep considering worst ranked candidates to maintain operators diversity. In [5], the authors have explored the advantages of using a controlled crossover on top of single-point search based hyper-heuristics. They maintain the best solutions obtained during the search and update crossover operator accordingly. The authors rely on a process focused on crossover as a biological selective breeding. This breeding assumes that fittest genes are already present in the initial population. Unlike cited approaches, Sputnik focuses on artificial mutation selection, therefore mimicking the process used to produce genetically modified organisms. As far as we know, there is no other hyper-heuristics that proposes artificial mutation at mutators level.

6. CONCLUSION

In this paper, we have introduced a novel hyper-heuristic breaking the random natural selection of classical MOEA to leverage instead an elitist artificial mutation inspired by biological studies [18] supported by a continuous learning of mutations impact. Sputnik, relies on a mutation operator selection based on a continuous learning of past effect on fitness functions. The overall goal of Sputnik is to enhance the optimization algorithm itself, and to guide the search towards faster trade-off achievement to finally save generation cycle and time. Integrated in a model-based optimization framework, our approach has been validated on the construction of a cloud reasoning engine prototype. Experimentally, we provide evidence of the effectiveness of artificial guided mutation to reduce significantly the number of necessary generations to find acceptable trade-offs. In the future, we plan to evaluate the relevancy of our heuristic on crossover operations. Also we plan to classify elitist strategies to analyze their impact on solution diversity.

7. REFERENCES

[1] G. Blair, N. Bencomo, and R. France. Models@ run.time.

[2] C. Darwin and J. Burrow. *The Origin of Species by Means of Natural Selection*. YNY, 1962.

[3] K. Deb. Multi-objective optimization. *Multi-objective optimization using evolutionary algorithms*.

[4] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *EC*, 2002.

[5] J. H. Drake, E. Özcan, and E. K. Burke. Controlling crossover in a selection hyper-heuristic framework. 2011.

[6] B. et al. A classification of hyper-heuristic approaches. In *Handbook of Metaheuristics*.

[7] F. et al. Search-based genetic optimization for deployment and reconfiguration of software in the cloud. In *ICSE*, 2013.

[8] F. F. et al. Dissemination of reconfiguration policies on mesh networks. In *DAIS*, 2012.

[9] I. A. G. et al. Hyper-heuristics for performance optimization of simultaneous multithreaded processors. In *ISCIS*, 2013.

[10] S. et al. Pareto rank learning in multi-objective evolutionary algorithms. In *CEC 2012*.

[11] Z. et al. The hypervolume indicator revisited. In *EMO*, 2007.

[12] D. Goldberg. Genetic algorithms in search, optimization, and machine learning, addison-wesley, reading, ma, 1989.

[13] J. H. Holland. *Adaptation in natural and artificial systems*. U Michigan Press, 1975.

[14] M. T. Jensen. Reducing the run-time complexity of multiobjective eas: The nsga-ii and other [...]. *EC*, 2003.

[15] C. León, G. Miranda, and C. Segura. Hyperheuristics for a dynamic-mapped multi-objective island-based model. 2009.

[16] S. W. Mahfoud. Niching methods for genetic algorithms. *Urbana*, 1995.

[17] E. Özcan, B. Bilgin, and E. E. Korkmaz. A comprehensive analysis of hyper-heuristics. *Intelligent Data Analysis*, 2008.

[18] H. Plaskett. Artificial transmutationof the gene. *TIC 1927*.

[19] C. R. Reeves and J. E. Rowe. *Genetic algorithms-principles and perspectives*. 2002.

[20] D. A. Van Veldhuizen. *Multiobjective evolutionary algorithms: classifications, analyses, and new innovations*. PhD thesis, 1999.

[21] D. A. Van Veldhuizen and G. B. Lamont. Multiobjective evolutionary algorithms: Analyzing the state-of-the-art. *Evolutionary computation*, 2000.

[22] B. Yuan and al. Statistical racing techniques for improved empirical evaluation of evolutionary algorithms. 2004.